Abstract Communication for Coordinated Planning

Bradley J. Clement
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 126-347
Pasadena, CA 91109-8099 USA
bclement@aig.jpl.nasa.gov

Edmund H. Durfee
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109-2110 USA
durfee@umich.edu

Abstract

Previous work offers evidence that distributed planning agents can greatly reduce communication costs by reasoning at abstract levels. While it is intuitive that improved search can reduce communication in such cases, there are other decisions about how to communicate plan information that greatly affect communication costs. This paper identifies cases independent of search where communicating at multiple levels of abstraction can exponentially decrease costs and where it can exponentially add costs. We conclude with a process for determining appropriate levels of communication based on characteristics of the domain.

Introduction

There are many motivations for distributed planning: physically separate points of control, shared resources, self-interested agents, computational constraints, and communication costs. This work demonstrates how communication costs can be reduced among a group of interacting planning agents by exploiting abstraction in hierarchical plans. The basic idea is that agents do not need to share their full plan information in the coordinated planning process. This can reduce communication delay or obtain a certain amount of privacy.

Hierarchical Task Network (HTN) planners (Erol, Hendler, & Nau 1994) represent abstract actions that decompose into choices of action sequences that may also be abstract, and HTN planning problems are requests to perform a set of abstract actions given an initial state. The planner subsequently refines the abstract tasks into less abstract subtasks to ultimately generate a schedule of primitive actions that is executable from the initial state. This differs from STRIPS planning where a planner can find any sequence of actions whose execution can achieve a set of goals. HTN planners only find sequences that perform abstract tasks and a domain expert can intuitively define hierarchies of abstract tasks to make the planner rapidly generate all sequences of interest.

Previous research (Korf 1987; Knoblock 1991) has shown that, under certain restrictions, hierarchical refinement search reduces the search space by an exponential factor. Subsequent research has shown that these restrictions can be dropped by reasoning during refinement about the conditions embodied by abstract actions (Clement & Durfee 2000; 1999; Clement *et al.* 2001). These *summarized conditions* represent the internal and external requirements and effects of an abstract action and those of any possible primitive actions into which it can decompose. Using this information, a planner can detect and resolve conflicts between abstract actions and sometimes can find abstract solutions or determine that particular decomposition choices are inconsistent.

In this paper, we examine how these abstract reasoning techniques affect communication. While it may seem obvious that less information is communicated at abstract levels, it is not clear when abstract information should be shared. Should agents send their entire plan hierarchies at once, only send subplans one-at-a-time as they are requested, or send levels of subplans? We show that it depends on the domain and describe a process for determining efficient communication strategies based on the domain.

We use a manufacturing domain to illustrate our approach. In the next section, we describe this domain, and then we give some background on summary information. We next analyze the communication costs of the manufacturing example and domains in general to show which strategies minimize communication costs. We then describe the process for determining good strategies.

Manufacturing Domain

The following simple example illustrates a particular problem to motivate this work. Consider a manufacturing plant where a production manager, a facilities manager, and an inventory manager each have their own goals and have separately constructed hierarchical plans to achieve them. However, they still need to coordinate over the use of equipment, the availability of parts used in the manufacturing of other parts, storage for the parts, and the use of transports for moving parts. The state of the factory is shown in Figure 1. In this domain agents can produce parts using machines M1 and M2, service the machines with a *tool*, and move parts

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.
Part of this work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. This work was also supported in part by DARPA(F30602-98-2-0142).

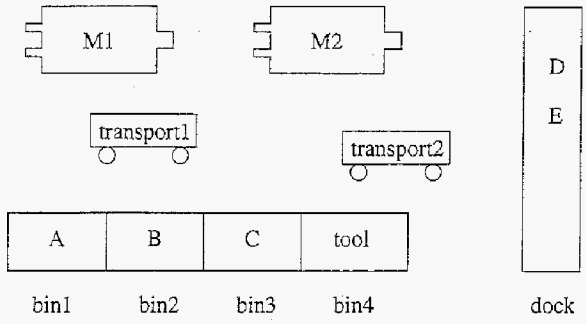

Figure 1: A simple example of a manufacturing domain

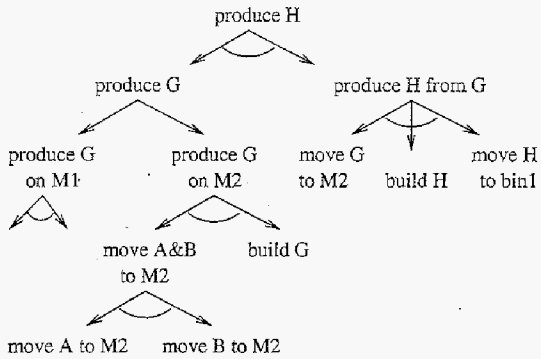

Figure 2: The production manager's hierarchical plan

to and from the shipping dock and storage bins on the shop floor using transports. Initially, machines M1 and M2 are free for use, and the transports (transport1 and transport2) and all of the parts (A through E) shown in their storage locations are available.

The production manager is responsible for creating a G and an H part using machines M1 and M2. Both M1 and M2 can consume parts A and B to produce G. M2 can also produce H from G. The production manager's hierarchical plan for manufacturing H involves using the transports to move the needed parts from storage to the input trays of the machines, manufacturing G and H, and transporting H back to storage. This plan is shown in Figure 2. Arcs through subplan branches mean that the conjunction of subplans must be executed. Branches without arcs mean that only one plan must be executed successfully to achieve the goal of the parent. The decomposition of $produce_G_on_M1$ is similar to that of $produce_G_on_M2$.

The facilities manager must service each of these machines by equipping it with a tool and then calibrating it. The machines are unavailable for production while they are being serviced. The facilities manager's hierarchical plan branches into choices of servicing the machines in different orders and uses the transports for getting the tool from storage to the machines (Figure 3). The decomposition of $service_M2M1$ is similar to that of $service_M1M2$.

The parts must be "available" on the space-limited shop floor in order for an agent to use the machines to produce them. Whenever an agent moves or uses a part, it becomes

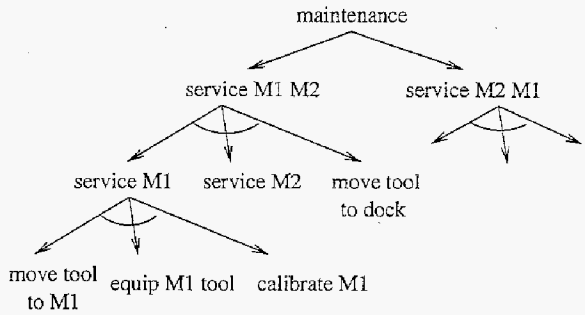

Figure 3: The facilities manager's hierarchical plan

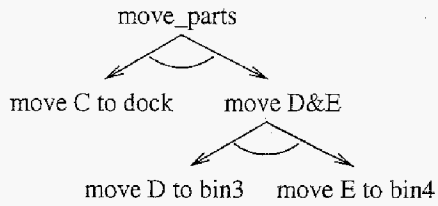

Figure 4: The inventory manager's hierarchical plan

unavailable. The inventory manager's goal is just to move part C to the dock and move D and E into bins on the shop floor (shown in Figure 4).

So what makes this a decentralized planning problem?

- The managers' plans use common resources, requiring them to coordinate or risk resource conflicts leading to failure of achieving their goals.
- The managers developed their own plans *a priori* and are ignorant of the intentions of the others.
- The managers are self-interested because they are separately accountable for their goals.

One might assume that it is unlikely that communication between workstations in a factory would introduce any significant overhead to the planning problem. We will show that there is significant overhead when the plans scale. This domain is based on operators with conditions and effects on propositional states. Later, we will expand on this domain with deeper hierarchies and metric resources.

Summary Information

In this section, we give a brief overview of summary information. For detailed information, consult (Clement & Durfee 1999; Clement *et al.* 2001; Clement 2002).

Summary information for a task represents the conditions and effects embodied in the task's potential decompositions. These conditions can be summarized for both propositional state and metric resources. An abstract task's state variable summary information includes elements for pre-, in-, and postconditions. Summary preconditions are conditions that must be met by the initial state or previous external tasks in order for a task to decompose and execute successfully, and a task's summary postconditions are the effects of its decomposition's execution that are not undone internally. We use summary inconditions for those conditions that are required

or asserted in the task's decomposition during the interval of execution. All summary conditions are used to reason about how state variables are affected while performing an abstract task, and they have two orthogonal modalities:

- *must* or *may* indicates that a condition holds in all or some decompositions of the abstract task respectively and
- *first*, *last*, *sometimes*, or *always* indicates when a condition holds in the task's execution interval.

Suppose that the factory managers in the example wish to coordinate their plans. Each agent could have preprocessed its plan to derive summary information for every abstract plan operator in the hierarchy. For a particular abstract plan, this information includes the summary pre-, post-, and in-conditions that correspond to the *external preconditions*, *external postconditions* and *internal conditions* respectively of the plan based on its potential refinements. Below I show a subset of state summary conditions for the production manager's top-level plan. Following each literal are modal tags for *existence* and *timing* information. "Mu" is *must*; "Ma" is *may*; "F" is *first*; "L" is *last*; "S" is *sometimes*; and "A" is *always*.

Production manager's *produce_H* plan:

Summary preconditions:

`available(A)MuF, available(M1)MaS, available(M2)MaS`

Summary inconditions:

`available(M1)MaS, available(M2)MuS, available(G)MuS, ¬available(A)MuS, available(A)MuS, ¬available(M1)MaS, ¬available(M2)MuS, ¬available(G)MuS, available(H)MuS, ¬available(H)MuS`

Summary postconditions:

`¬available(A)MuS, available(M1)MaS, ¬available(G)MuS, available(M2)MuS, available(H)MuL`

$available(M2)$ is a *must* precondition of the top-level plan, $produce_H$, because no matter how the hierarchy is decomposed, M2 must be used to produce H, so $available(M2)$ must be established externally to the production manager's plan. However, $available(M1)$ is a *may* precondition because the production manager may not use M1 if it chooses to use M2 instead to produce G. $available(A)$ is a *first* summary precondition because part A is needed at the beginning of execution when it is transported to one of the machines. Since the machines are not needed at the very beginning when parts are being transported, they are not *first* but *sometimes* conditions since they are needed at some point in time during execution.

A *summarized resource usage* consists of ranges of potential resource usage amounts during and after performing an abstract task, and we represent this summary information using the structure

$$\langle local_min_range, local_max_range, persist_range \rangle,$$

where the resource's local usage occurs within the task's execution, and the persistent usage represents the usage that lasts after the task terminates for depletable resources.

The usage ranges capture the multiple possible usage profiles of an task with multiple decomposition choices and timing choices among loosely constrained subtasks. For example, in the manufacturing domain suppose A, B, G, etc. are groups of parts represented as metric resources, as well as the transports; the storage bins and dock have limited capacity for parts; and the machines consume and produce numbers of parts at a time based on the number of input parts. If the production manager's plan was to produce 20 H parts, the summarized resource usage would the following:

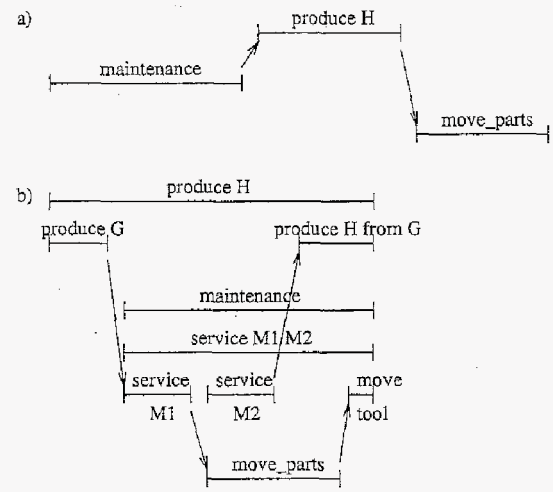

Figure 5: Solutions to the manufacturing domain

A	$\langle[-20,-20],[0,0],[-20,-20]\rangle$
G	$\langle[0,0],[20,20],[0,0]\rangle$
H	$\langle[0,0],[20,20],[20,20]\rangle$
M1	$\langle[0,0],[1,1],[0,0]\rangle$
M2	$\langle[0,0],[0,1],[0,0]\rangle$

The usages for A mean that over the duration of the top-level plan, the quantity of A will definitely be depleted [-20,-20], but there is at least some period where A is not consumed [0,0]. G is produced and consumed, so there are no persistent effects. M1 and M2 are non-depletable (atomic) resources, so they also have no persistent effects.

With summary information derived for the managers' plan hierarchies, the production and inventory managers could send the summary information for their top-level plans to the facilities manager. The facilities manager could then reason about the top-level summary information for each of their plans to determine that if the facilities manager serviced all of the machines before the production manager started producing parts, and the production manager finished before the inventory manager began moving parts on and off the dock (as shown in Figure 5a), then all of their plans can be executed (refined) without conflict. Then the facilities manager could instruct the others to add communication actions to their plans so that they synchronize their actions appropriately.

By reasoning at abstract levels with summary information, planning agents can find solutions at abstract levels, prune inconsistent decompositions, and guide the decomposition to quickly resolve conflicts. This results in a doubly exponential speedup in planning time (Clement & Durfee 2000; Clement *et al.* 2001). Next we explore how this affects communication.

Communication Experiments

Here we show that, depending on how communication costs are calculated and how summary information is communicated among the agents, communication costs may be exponentially reduced. If only the number of messages is a concern, then at one extreme sending the plan hierarchy without summary information makes the most sense. At another extreme it makes sense to send the summary information for each task in a separate message as each is requested. Still, there are cases when sending the summary information for tasks in groups makes the most sense. This section will explain how a system designer can choose an appropriate level of granularity to send summary information, potentially reducing communication overhead exponentially.

Consider a simple protocol where agents request coordination from a central coordinating agent. During the search for a feasible solution, whenever it decomposes a task, the coordinator requests summary information for the subtasks that it has not yet received. For the manufacturing domain, the coordinator may already have summary information for a task to move a part, but if it encounters a different instantiation of the same task schema, it still must request the parameters for the new task. If a coordinator needs the subplans of an *or* plan, the client agent sends the required information for all subplans specifying its preferences for each. The coordinator then chooses the most preferred subplan, and in the case it must backtrack, it chooses the next most preferred subplan. Once the coordinator finds a feasible solution, modifications are sent to each agent specifying which *or* subplans are blocked and where it must send and wait for synchronization messages. An agent can alternatively send summary information for the whole plan hierarchy up front, for single tasks as they are requested, or for some number of levels of expansion of the requested task's hierarchy.

Communication cost is often measured in terms of number of messages, total amount of data, or a weighted function of the two. Communication delay is one such function and can be computed as $n\ell + s/b$, where n is the number of messages sent; ℓ is the latency in seconds; s is the total size of all messages; and b is the bandwidth in bytes per second. We will use this function in our manufacturing examples.

Simple Manufacturing Problem

For the simple manufacturing problem first described with propositional states, communication data in terms of numbers of messages and the size of each was collected up to the point that the coordinator found the solution in Figure 5b. This data was collected for cases where agents sent summary information for tasks in their hierarchies one at a time, two levels at a time, and all at once. The two levels include the requested task and its immediate subplans. The following table below summarizes the numbers and total sizes of messages sent for each granularity level of information:

Simple manufacturing example results

	number of messages	total size (bytes)
one at a time	9	8708
two levels	4	10525
all at once	3	16268

Assuming that the coordinator must wait for requested information before continuing its search and can request only one task of one agent at a time, the coordination will be delayed for an amount of time depending on the bandwidth and latency of message passing. The total delay can be calculated as $(n-2)\ell + s/b$. We use $n-2$ instead of n because we assume that the agents all transmit their first top-level summary information message at the same time, so three messages actually only incur a delay of ℓ instead of 3ℓ.

Figure 6 shows how the communication delay varies for the three granularities of information transfer for a fixed bandwidth of 100 bytes/second. When the latency is less than 0.003 seconds, sending summary information for each task in separate messages results in the smallest communication overhead. For latencies greater than 0.06 seconds, sending the entire hierarchy is best; and in between sending summary information two levels at a time is better. Fixing the latency at 0.01 seconds, we see the boundaries between the strategies in Figure 7 where communication delay varies with bandwidth. When the bandwidth is less than 37 Kbytes/second, sending one at a time is best. Sending it all at once is only best for very high bandwidths. For moderately high bandwidths, sending two levels at a time is best.

However, no matter how data is sent, only a half second could be saved at most, which is certainly insignificant for the domain. This simple scheduling problem was solved using 1.23 cpu seconds, so the difference is actually a significant percentage. More realistic problems involving the manufacturing domain could have much larger hierarchies and require much larger amounts of data to be sent and communication overhead.

Larger Example Problem

We next coordinate three production managers with the facilities and inventory manager. All agents schedule using metric resources as described in the section on Summary Information. In this domain, there are three transports, six machines, and seven part types. Some parts are raw materials, others have combination rules for production, and machines are capable of producing different subsets of parts. For example, for $3x$ number of C and $2x$ D parts, x E parts can be produced.

The production managers have plans to produce two batches of parts with optional decompositions (e.g. produce 20 C and either 30 E or 10 G). Their plans also branch on machine choices. Although there is only moderate branching, the hierarchy depth is 12, and there are 14 leaf actions. The inventory manager move four loads of parts back and forth from storage and the dock. The facilities manager must service all machines in any order. The inventory manager's plan has a depth of 12 with 42 leaves. The facilities manager's plan has a depth of 9 and 12 leaves. All leaf actions use shared resources.

The coordinator resolves conflicts in 2.8 to 3.2 cpu seconds. The resulting solution is found at about half the depth of the agents' hierarchies. The following table lists the number of messages sent and the total data transfer size for each communication case. (Note that protocol for transmitting summary information does not affect search—it only delays

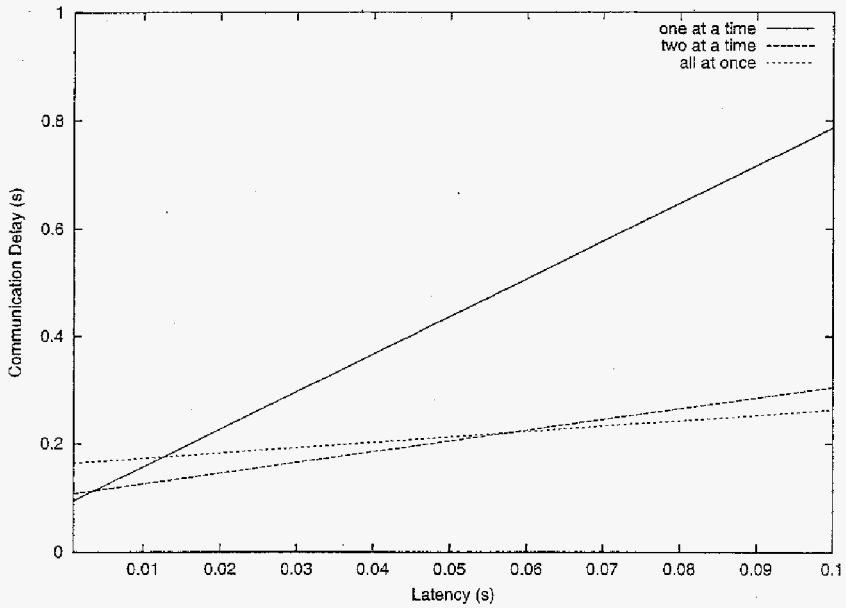

Figure 6: Simple problem: delay of communicating at different granularities with varying latency

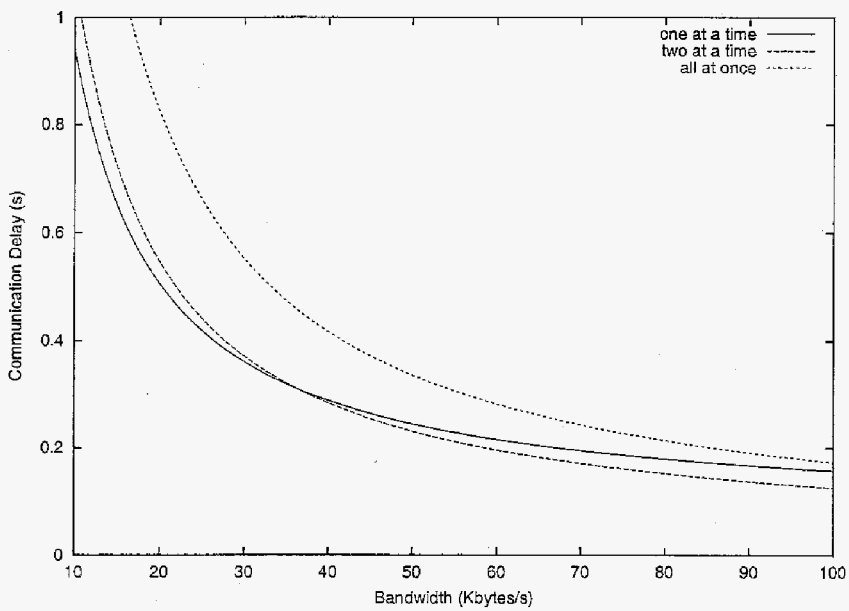

Figure 7: Simple problem: delay of communicating at different granularities with varying bandwidth

it). About 10% of the data transmission results from backtracking.

Larger manufacturing example results

	number of messages	total size (Kbytes)
one at a time	52	55.22
two levels	15	73.43
all at once	5	527.34

The comparisons for varying bandwidth and latency are shown in Figures 8 and 9 fixing bandwidth to 100 Kbytes/s and latency to 0.01 s. Here, we see that if the latency is small, there is no significant difference between sending each action separately or sending two levels of the hierarchy in each message. However, the intermediate granularity approach saves three to more than 10 seconds compared to sending the entire hierarchies at the start (for the plotted ranges). This completely dominates the cpu processing. While 10 seconds may not be important for scheduling hours of factory operation, clearly the problems can be much larger and involve greater decentralized decision making in backtracking, resulting in retransmitting parts of the hierarchy.

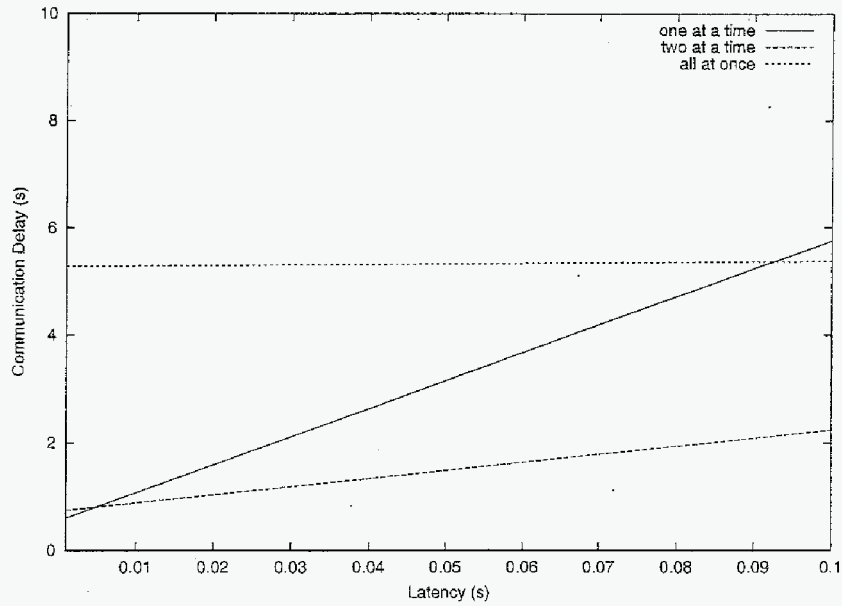

Figure 8: Larger problem: delay of communicating at different granularities with varying latency

Analysis and Discussion

For three agents, a branching factor of 3, depth 10, a solution found at level 5; and 1 Kbyte of summary information for any task, the table would look like this:

Hypothetical case

	number of messages	total size (Kbytes)
one at a time	363	1.01
two levels	9	3.28
all at once	3	246.03

Without considering a particular communication cost function, it is evident that sending plan information at intermediate levels of abstraction will often provide a more efficient alternative to the extremes represented by the first and last rows of the table.

Suppose that m agents' hierarchies had a common branching factor b and depth d. If tasks generally had constraints on similar resources throughout the hierarchies, the amount of summary information derived for the tasks at particular levels would grow exponentially down the hierarchy just as would the number of tasks (Clement *et al.* 2001). If the agents agreed on a feasible solution at depth level i in the hierarchy, then the table for messages and size would appear as follows:

General case

	number of messages	total size
one at a time	$O(mb^i)$	$O(mb^i)$
k levels	mi/k	$O(mb^{i+k-1})$
all at once	$O(m)$	$O(mb^d)$

The first row shows that as the hierarchies scale, the number of messages grows exponentially when sending task information as it it requested. The last row shows that the total data sent grows exponentially. The middle row shows that the exponent on message number can be avoided, and the exponent on the total data can be mitigated when k is chosen small compared to d. The exponent is $i + k - 1$ because there is a risk of sending more information than is required. This is shown in the previous table where $k = 2$ forcing six levels (5+2-1) of data to be sent when the solution is found only five levels deep. Note that if an agent sends k levels of the hierarchy, only the deepest level needs to be summarized because the other levels can be summarized by the receiving agent.

These observations assume that the size of summary information grows exponentially at lower levels of the hierarchy along with the numbers of tasks. As discussed in (Clement *et al.* 2001), this happens when different tasks in the hierarchy have constraints on similar resources causing constraints to collapse during summarization. If they do not, then summary information does not shrink at higher levels—the data size is $O(b^d)$ at each level, so all messages are size $O(b^d)$:

Summary information does not collapse up hierarchy

	number of messages	total size
one at a time	$O(mb^i)$	$O(mib^d)$
k levels	mi/k	$O((mi/k)b^d)$
all at once	$O(m)$	$O(mb^d)$

In this case communication is minimized by sending the unsummarized hierarchy at the beginning of coordination. (The agent(s) receiving the plans can summarize them for planning efficiency once they are received.

Another assumption is that the solution will be found at an intermediate level of decomposition. If acceptable solutions are usually found near the top level, then summary information should only be sent when requested:

Solutions found at top-level

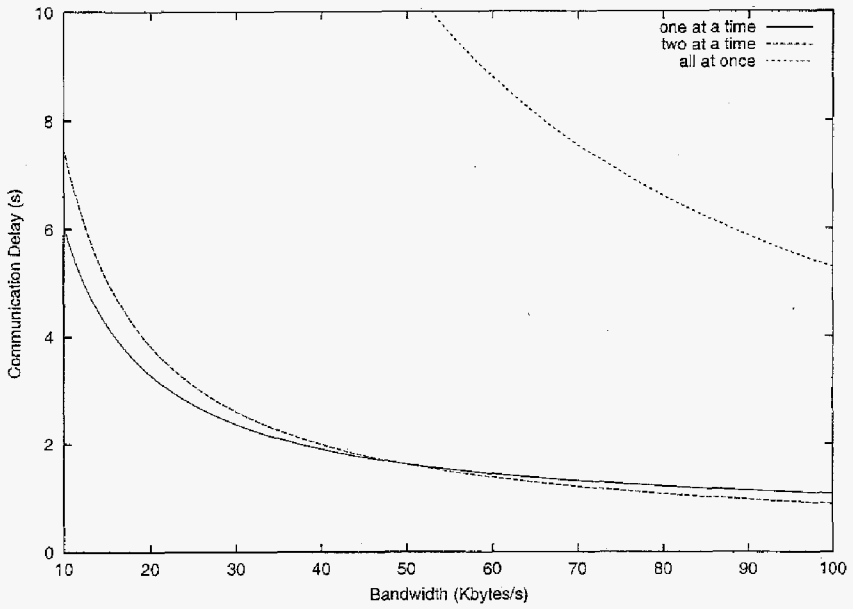

Figure 9: Larger problem: delay of communicating at different granularities with varying bandwidth

	number of messages	total size
one at a time	$O(m)$	$O(m)$
k levels	$O(m)$	$O(mb^k)$
all at once	$O(m)$	$O(mb^d)$

If acceptable solutions are only found at the bottom of the hierarchy, then sending summary information will increase overhead, so the hierarchies should be sent at the beginning:

Solutions found at bottom-level

	number of messages	total size
one at a time	$O(mb^d)$	$O(mb^d)$
k levels	$O(md/k)$	$O(mb^d)$
all at once	$O(m)$	$O(mb^d)$

The following procedure based on domain testing can be used for determining how to send summary information efficiently. The output is k, the granularity (number of levels) of communicating details of the hierarchy. This can be determined separately for each agent since the depth of solution in one agent's hierarchy could be independent from those of others.

- if summary information does not collapse up the hierarchy or
- if solutions only exist at the bottom of the hierarchy,
  - $k = d$.
- else if solutions are always found at the top-level or
- if the cost is only determined by the total data size,
  - $k = 1$.
- else if the cost is only determined by the number of messages,
  - $k = d$.
- else
  - pick k small with respect to d, and
  - experiment with variations of k in simulation and assign best.

The last case is only a "strawman" suggestion. Certainly, there are other schemes alternative to sending a fixed number of levels. For example, decreasing k during search could minimize the risk of sending levels deeper than the desired solution. k might be tailored to the mean and variance of solutions or many other factors, including the cost function.

Conclusion

It is intuitive that agents that reason about their plans at abstract levels should communicate less than those that reason at detailed levels in their plans. Instead of merely demonstrating that point, this paper identified how communicating abstract information at different granularities can make an exponential difference in performance. We showed through an example of a simple manufacturing domain that communication delay in modern networks can be the performance bottleneck for plan coordination if plan information is not communicated at appropriate levels. We presented a method for choosing appropriate levels based on domain characteristics.

References

Clement, B., and Durfee, E. 1999. Theory for coordinating concurrent hierarchical planning agents. In *Proc. AAAI*, 495–502.

Clement, B., and Durfee, E. 2000. Performance of coordinating concurrent hierarchical planning agents using summary information. In *Proc. ATAL*, 213–227.

Clement, B.; Barrett, A.; Rabideau, G.; and Durfee, E.

2001. Using abstraction in planning and scheduling. In *Proc. ECP*, 145–156.

Clement, B. 2002. *Abstract Reasoning for Multiagent Coordination and Planning*. Ph.D. Dissertation, University of Michigan, Ann Arbor.

Erol, K.; Hendler, J.; and Nau, D. 1994. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, University of Maryland.

Knoblock, C. 1991. Search reduction in hierarchical problem solving. In *Proc. AAAI*, 686–691.

Korf, R. 1987. Planning as search: A quantitative approach. *Artificial Intelligence* 33:65–88.